

LEARN

DRINK

JAVA™
TECHNOLOGY

LIVE PLAY



Using the Java™ Cryptographic Architecture (JCA) and Java™ Cryptography Extensions (JCE)

Steve Burnett
RSA Data Security, Inc.
burnett@rsa.com

Topics

- **What is the Java™ Cryptographic Architecture (JCA) and Java™ Cryptography Extensions (JCE)**
- **The JCA Programming Model**
- **Examples**
- **Random Number Generation**
- **[appendix]**

What Is JCA/JCE?

**JCA: Java™ Cryptographic Architecture,
Philosophy of cryptographic
API, some “implementation”**

**JCE: Java™ Cryptography Extensions,
Additions to the “implementation”
(export restrictions)**

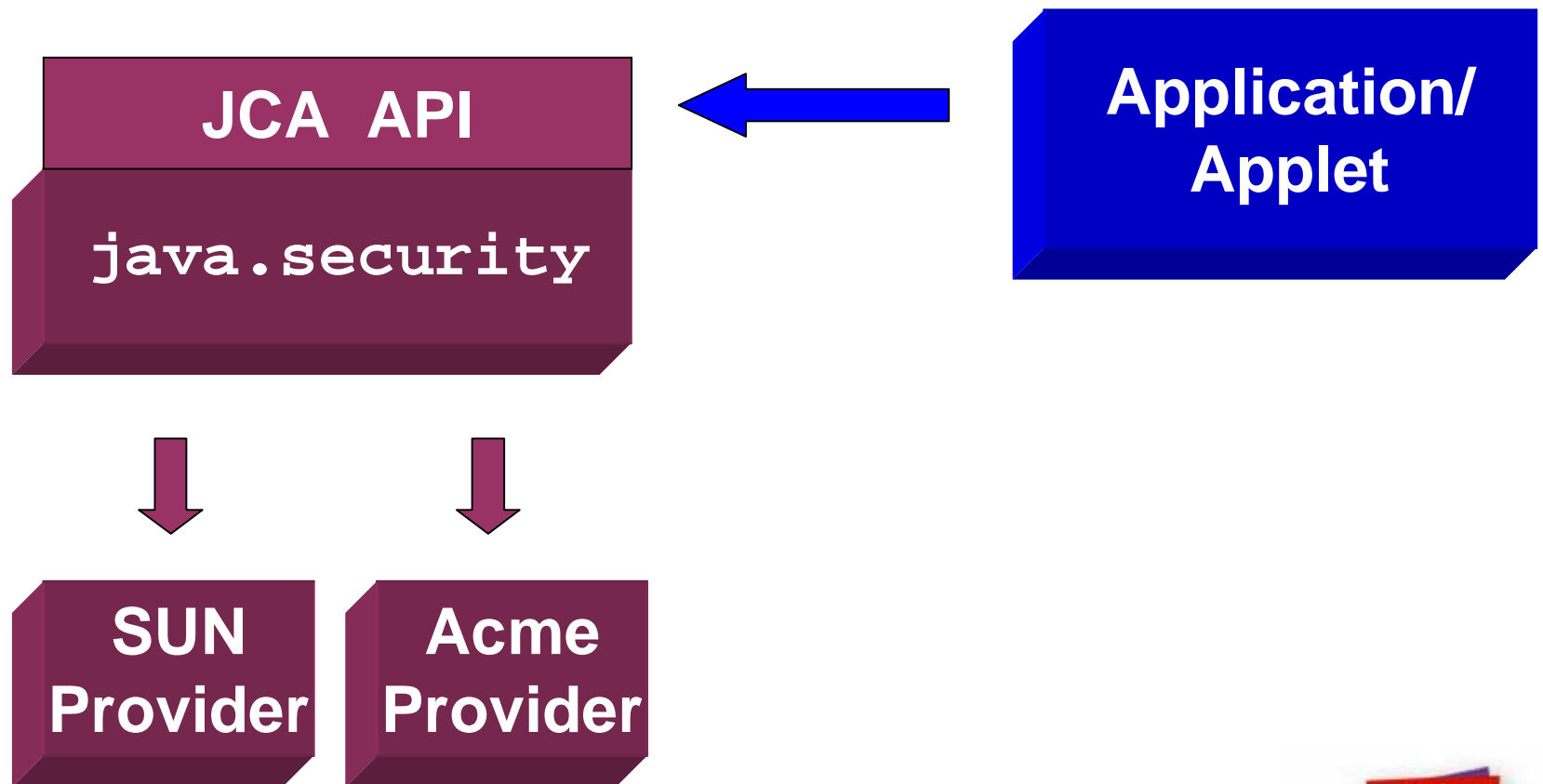


JCA

**Introduced MessageDigest, Signature,
SecureRandom classes
(subsidiary classes: keys, etc.)**

**Introduced the concept of Provider
(getInstance)**

JCA



JCE

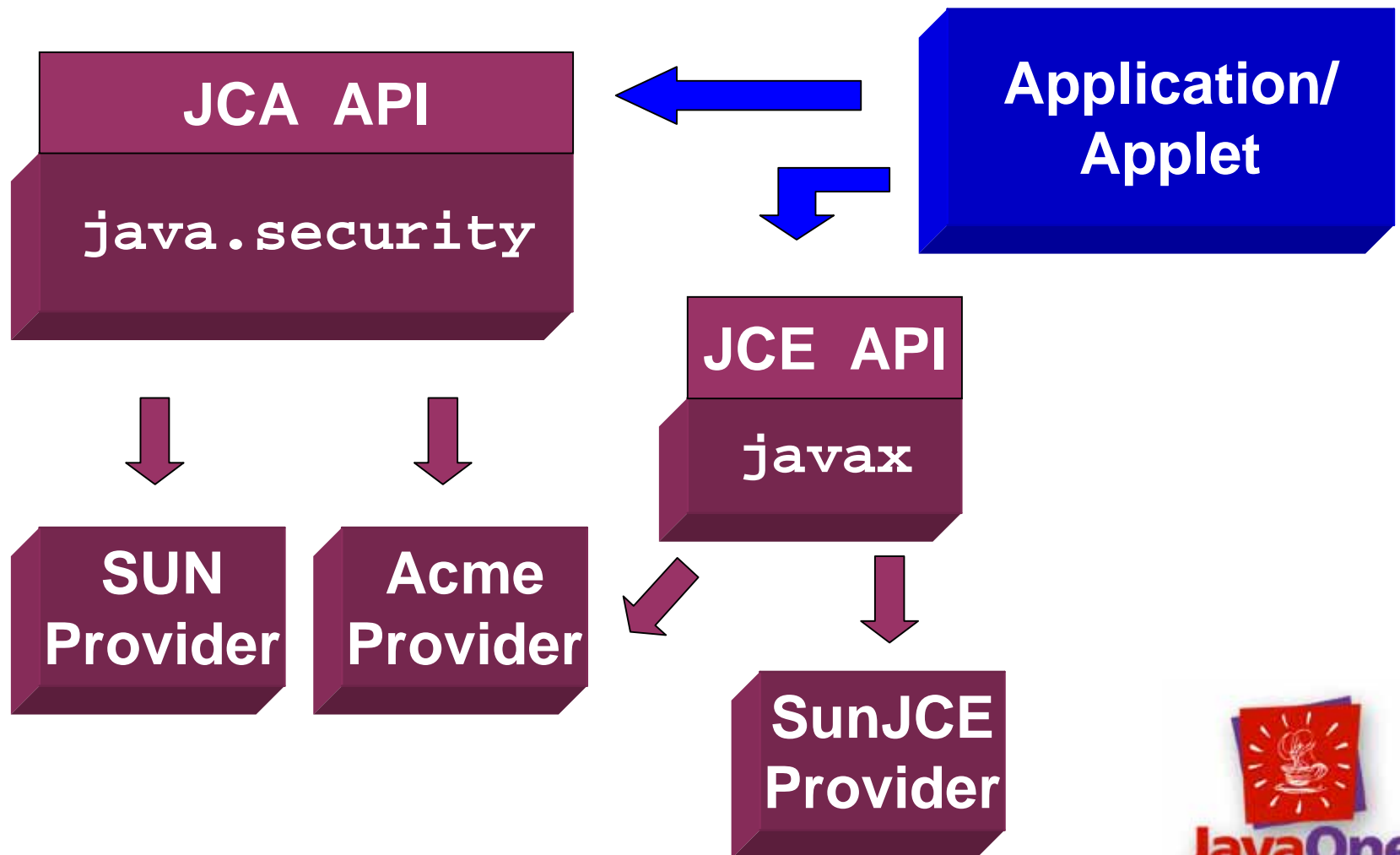
Extensions to the JCA

**Separate package due to
export restrictions**

**Adds Cipher, KeyExchange,
MAC classes
(subsidiary classes: keys, etc.)**



JCA/JCE



History

JCA introduced JDK™ 1.1 release

JCE distributed as “early access” at the same time

JCE early access withdrawn, never became a “product” during the JDK™ 1.1 timeframe



History

Why withdrawn?

Inadequate—For instance, could not store keys or make key objects from existing key data

DSA flaw

History

**JCA reworked for the JDK™ 1.2 release
(retaining backwards compatibility)**

**JCE rebuilt from “scratch” to be released
at the same time**

Advantages

- **Standard API**
- **You have some cryptographic functionality without having to go out and buy it**
- **Users and your customers will have this code, so you do not have to include the crypto as part of your product**

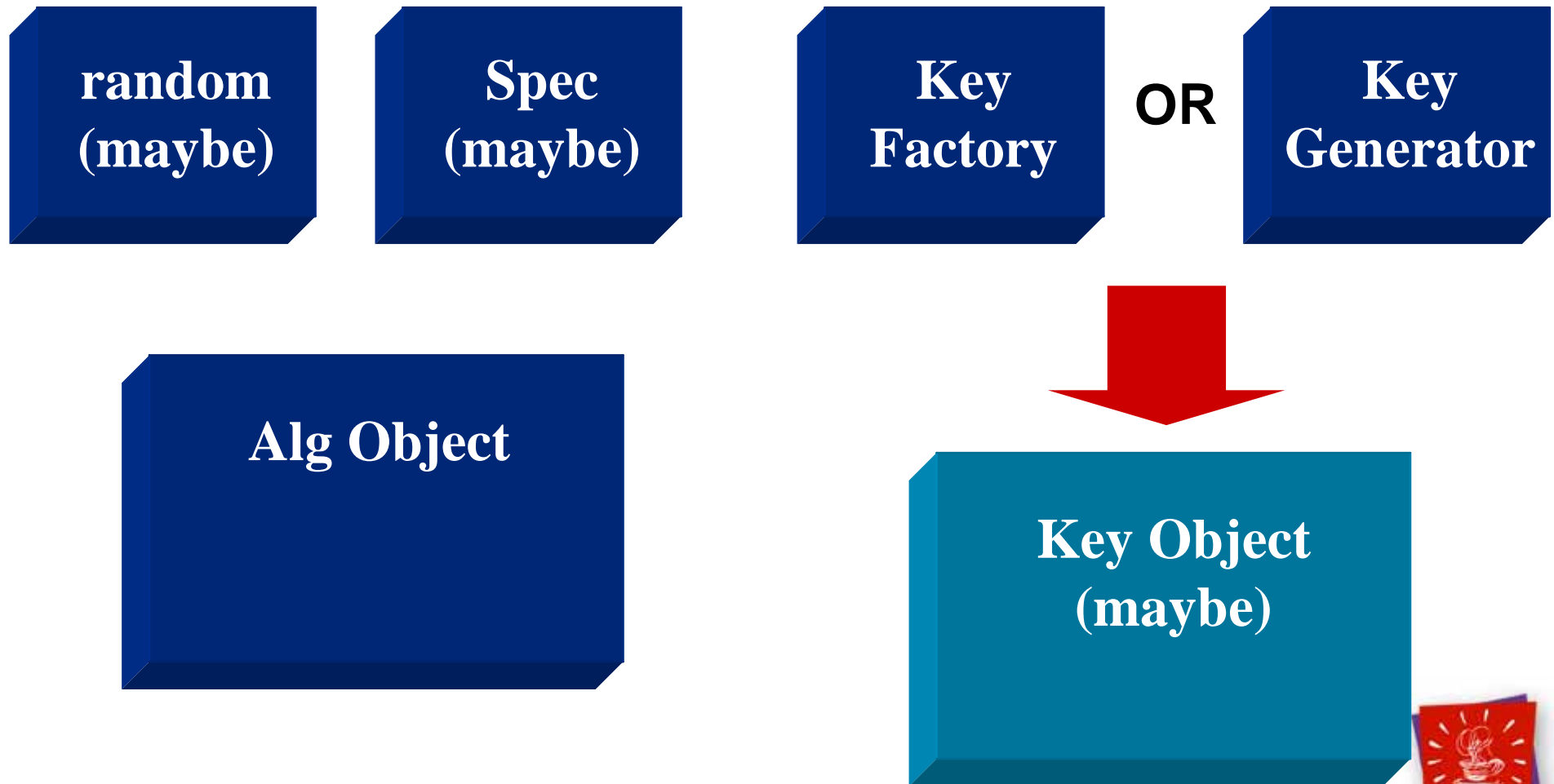
Disadvantages

- Java 2 platform (time to adopt)
- Missing algorithms (RSA, RC2, RC4)
- Hard to use (eight packages, many methods, complicated paths)
- Inconsistent (init vs. initialize vs. initSign vs. no init, getInstance vs. constructors)
- Real-world limitations (e.g. BER)

Topics

- What is the JCA/JCE
- The JCA Programming Model
- Examples
- Random Number Generation
- [appendix]

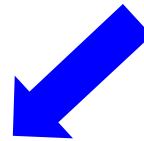
JCA/JCE Model



Factory vs. Generator

**Key
Factory**

**Key
Generator**

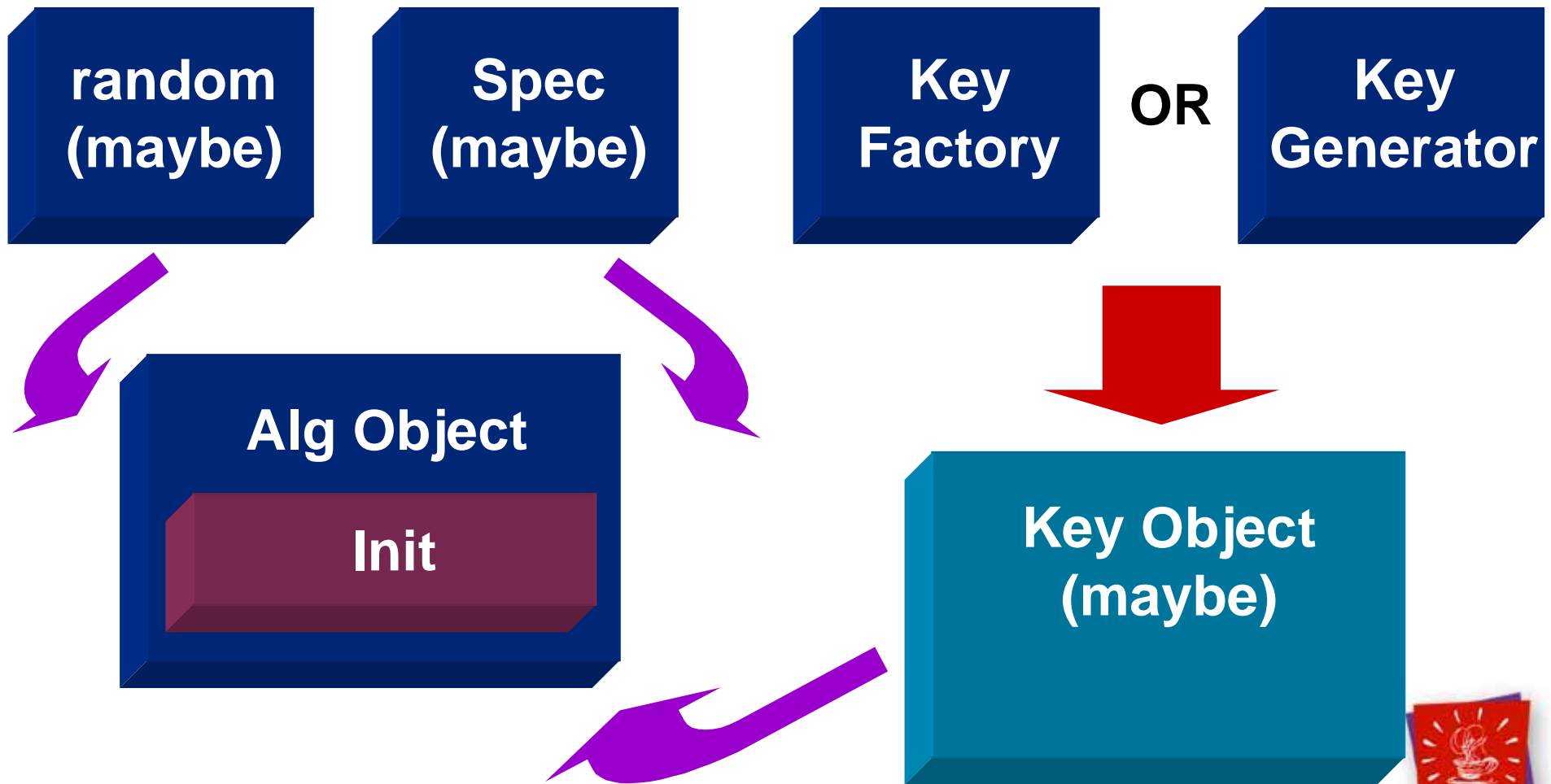


**Build a key object
from existing data**

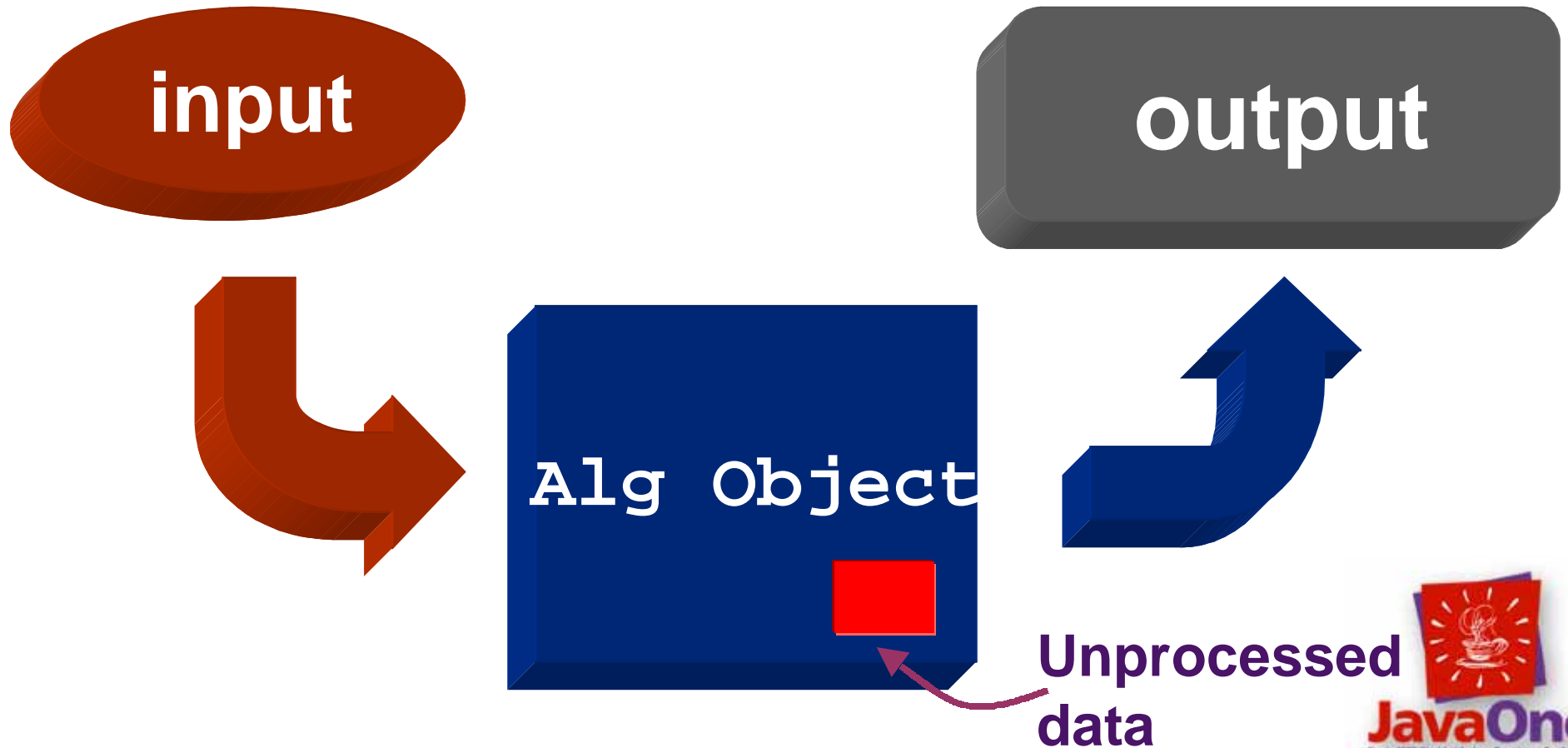


**Build a new key
from random data**

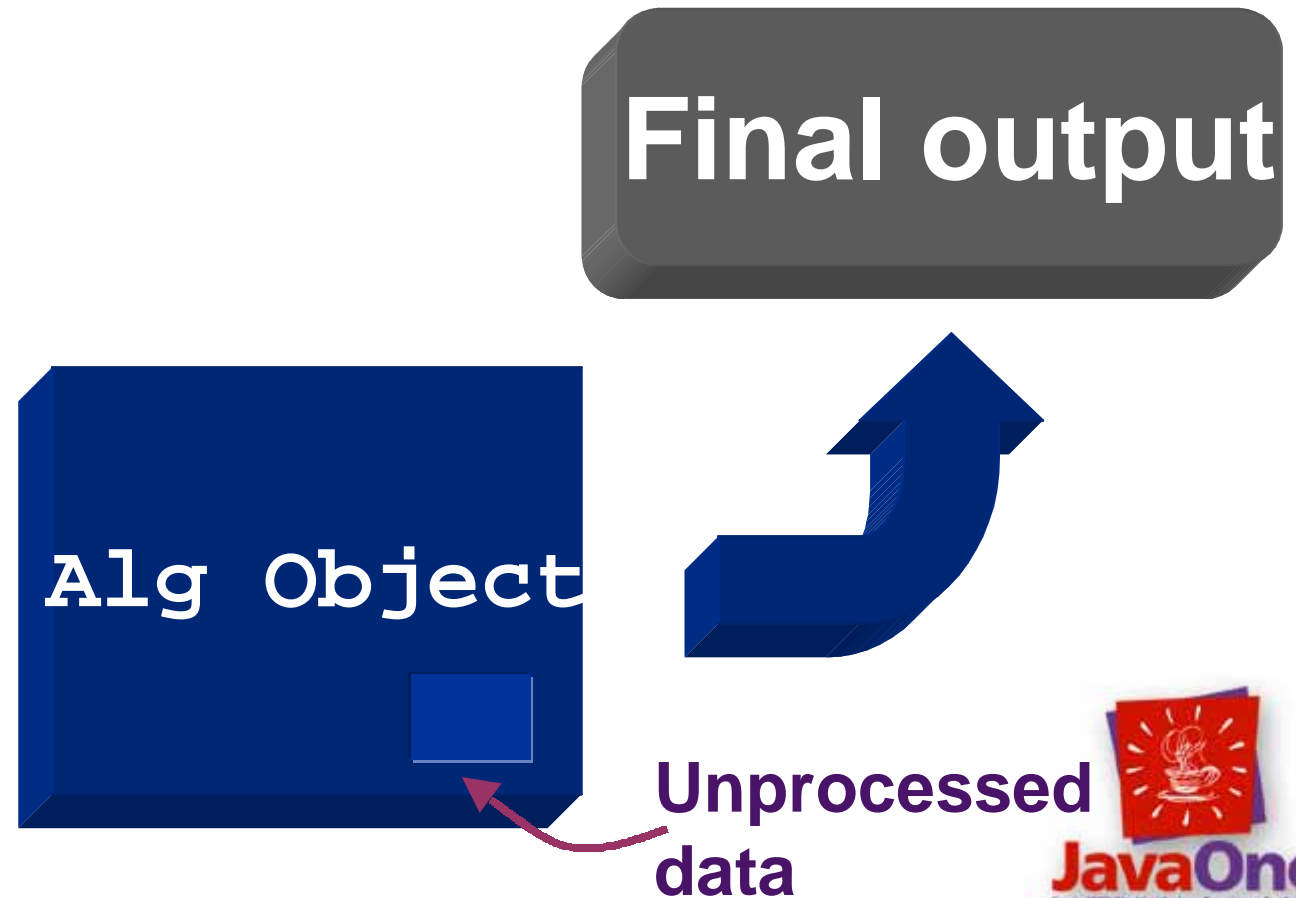
JCA/JCE Model



JCA/JCE Model



JCA/JCE Model



The JCA/JCE Model

1. **getInstance**
2. **init**
3. **update**
4. **final**

Topics

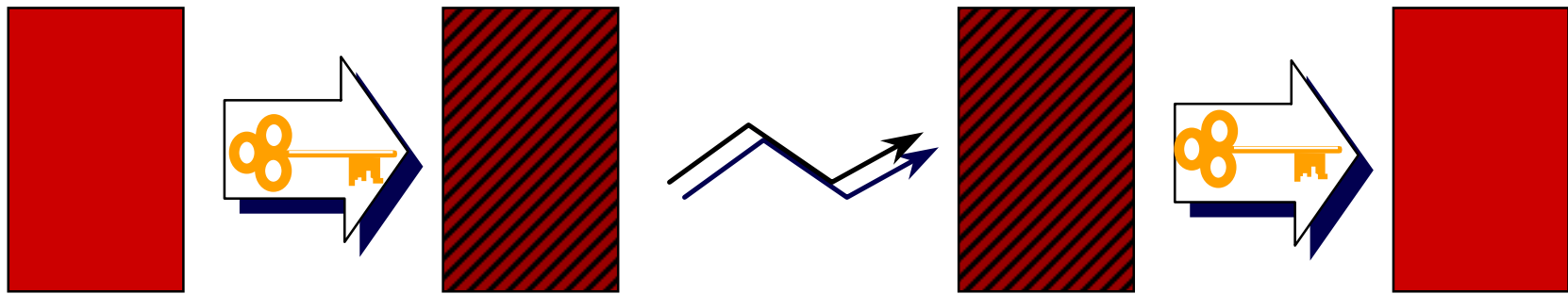
- What is the JCA/JCE
- The JCA Programming Model
- Examples
- Random Number Generation
- [appendix]

JCA/JCE Model Example

Digital Envelope

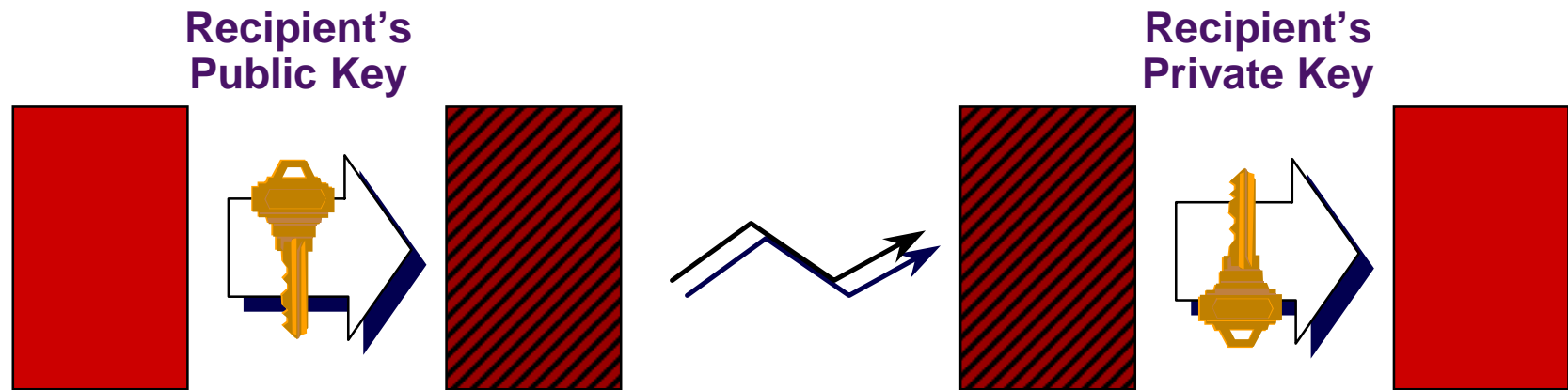
**Bulk encryption using a symmetric cipher (DES),
encrypt the symmetric key using RSA**

Symmetric vs. Public Key



Symmetric key cryptography: The same key used to encrypt is necessary to properly decrypt.

Symmetric vs. Public Key



Public key cryptography: Two keys, what one encrypts, the other decrypts. One is made public, the other remains private. Knowledge of the public key will NOT assist in breaking the message or system.

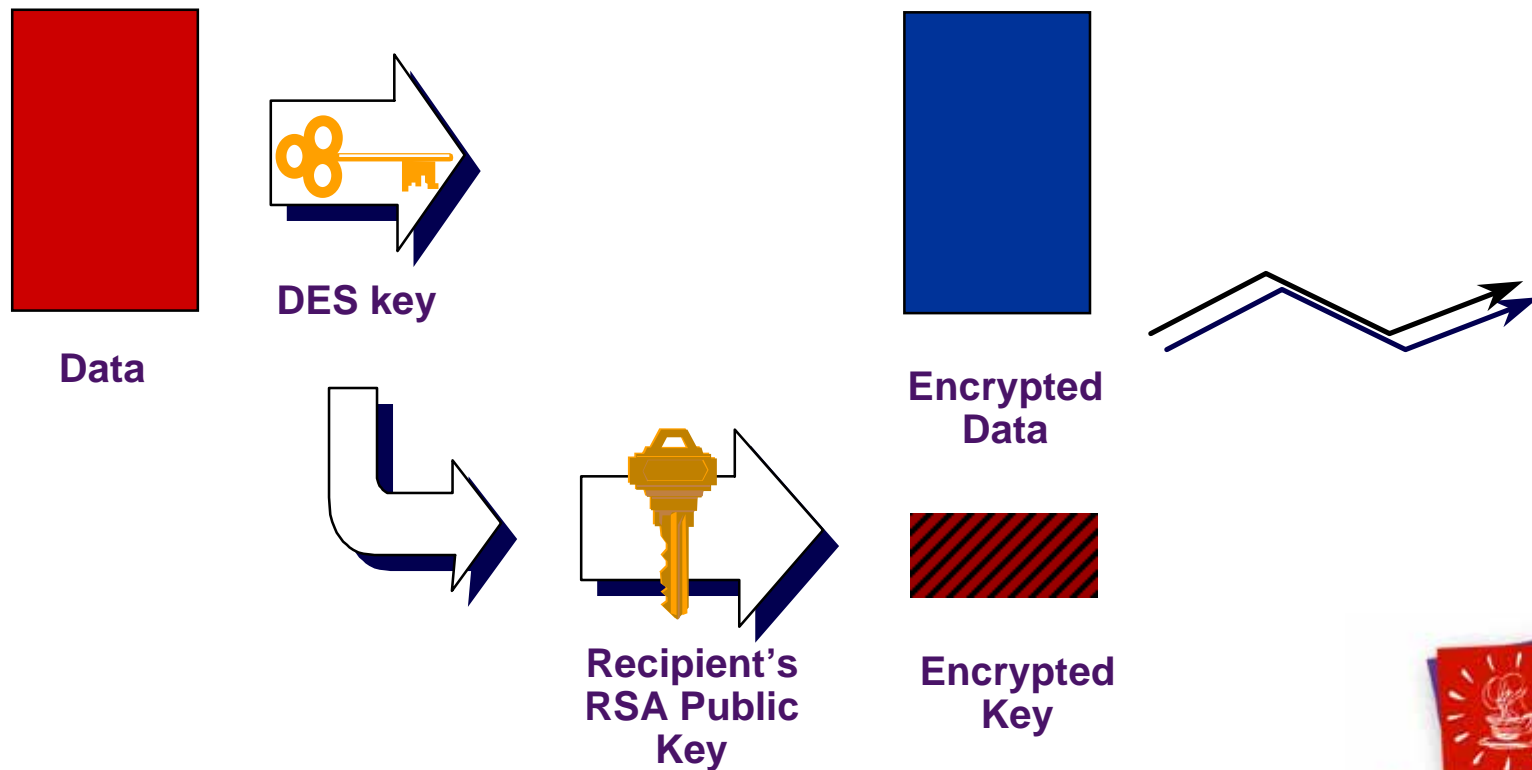
Symmetric vs. Public Key

Public key cryptography slow

Use symmetric encryption for the data, then encrypt the key (a smaller amount, generally 8 or 16 bytes) using public key

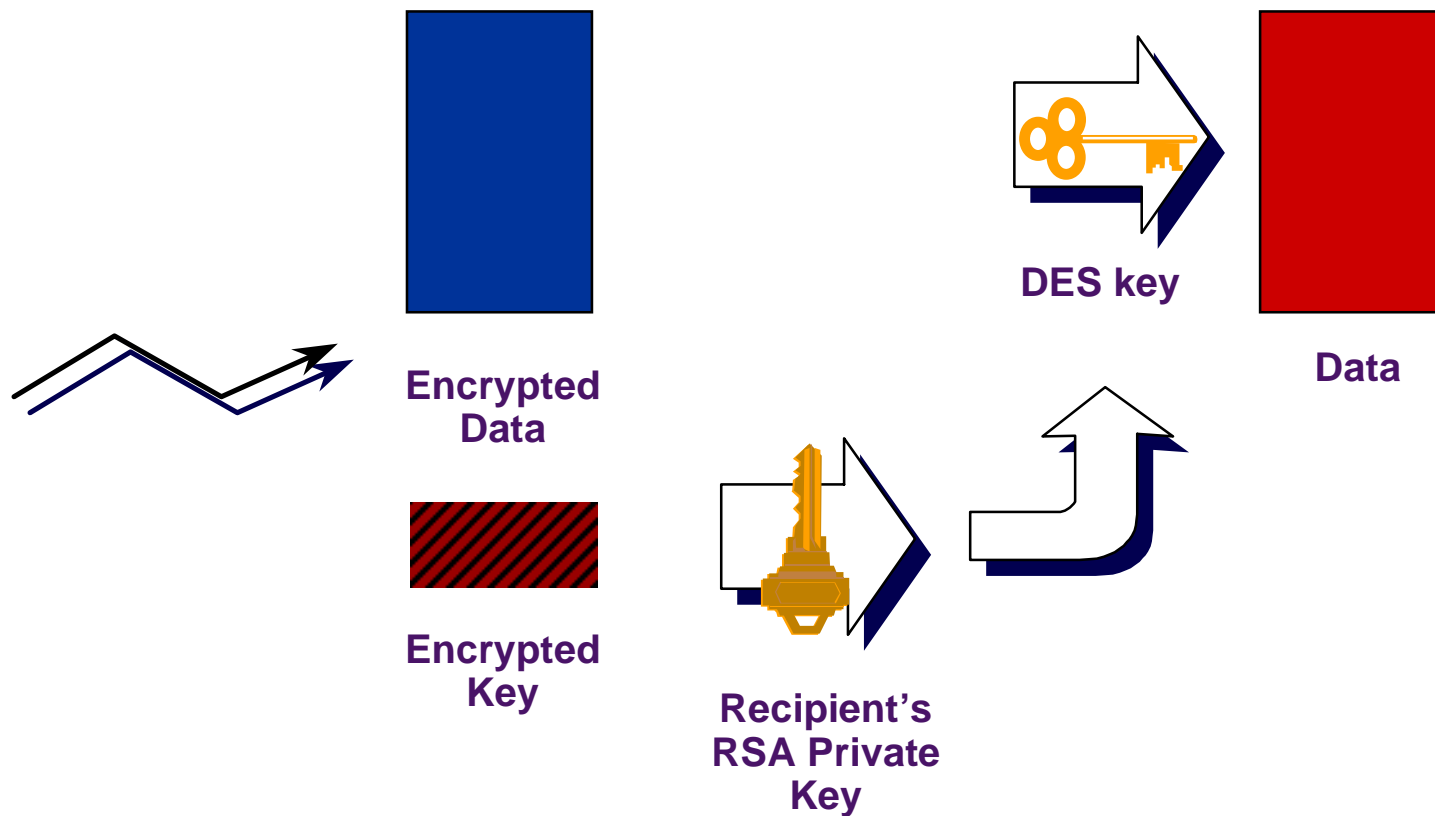
RSA Digital Envelope

Encrypt

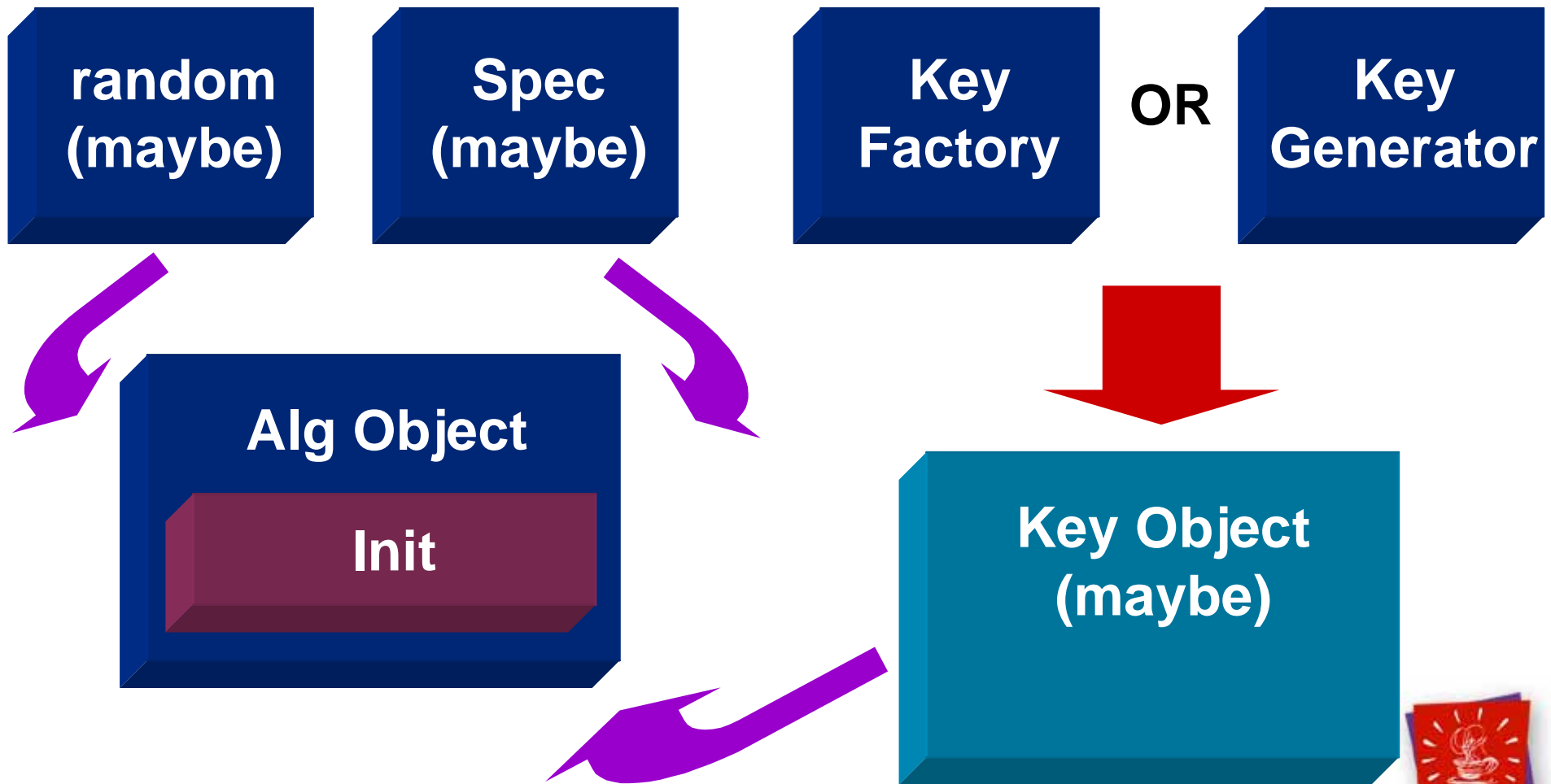


RSA Digital Envelope

Decrypt



JCA/JCE Model



DES With CBC

- Algorithm object: Cipher
- Spec (for IV): IvParameterSpec
- random (not needed)
- Key Object: SecretKey
(built from KeyGenerator)

Spec

```
SecureRandom random =  
    SecureRandom.getInstance (SHA1PRNG);  
random.setSeed  
    (System.currentTimeMillis ());  
  
byte[] iv = new byte [8];  
  
random.nextBytes (iv);  
  
IvParameterSpec desCBCParams =  
    new IvParameterSpec (iv, 0, 8);
```

Seeding a PRNG

**Generally, time of day is
not a complete seed**

**For an initialization vector,
something that will not be
kept secret, it is good enough**

Key Generator

- Algorithm object: KeyGenerator
- Spec (not needed)
- random: well-seeded random object
- Key Object (not needed—we're building a key)

KeyGenerator

Step 1: getInstance

```
Provider sunJCE = new  
    com.sun.crypto.provider.SunJCE ();  
Security.addProvider (sunJCE);
```

```
KeyGenerator desKeyGen =  
    KeyGenerator.getInstance  
        ( "DES", "SunJCE" );
```


Statically Loaded Provider

```
... jdk\jre\lib\java.security
```

```
security.provider.1 =  
    sun.security.provider.Sun
```

```
security.provider.2 =  
    com.sun.crypto.provider.SunJCE
```



Statically Loaded Provider

```
KeyGenerator desKeyGen =  
    KeyGenerator.getInstance  
        ( "DES" );
```

KeyGenerator

Step 2: init

```
byte[] newSeed =  
    random.generateSeed (20);  
random.setSeed (newSeed);
```

```
byte[] newSeed =  
    mySeedGenerator.getSeed (20);  
random.setSeed (newSeed);
```

```
desKeyGen.init (random);
```

KeyGenerator

Step 3, Step 4: Update, Final

```
SecretKey desKey =  
    desKeyGen.generateKey ();
```

DES With CBC

- Algorithm object: Cipher
- ✓ • Spec (for IV): IvParameterSpec
- ~~• random (not needed)~~
- ✓ • Key Object: SecretKey (built from KeyGenerator)

Cipher

Step 1: getInstance

```
Cipher des =  
    Cipher.getInstance  
        ( "DES/CBC/PKCS5Padding",  
          "SunJCE" );
```

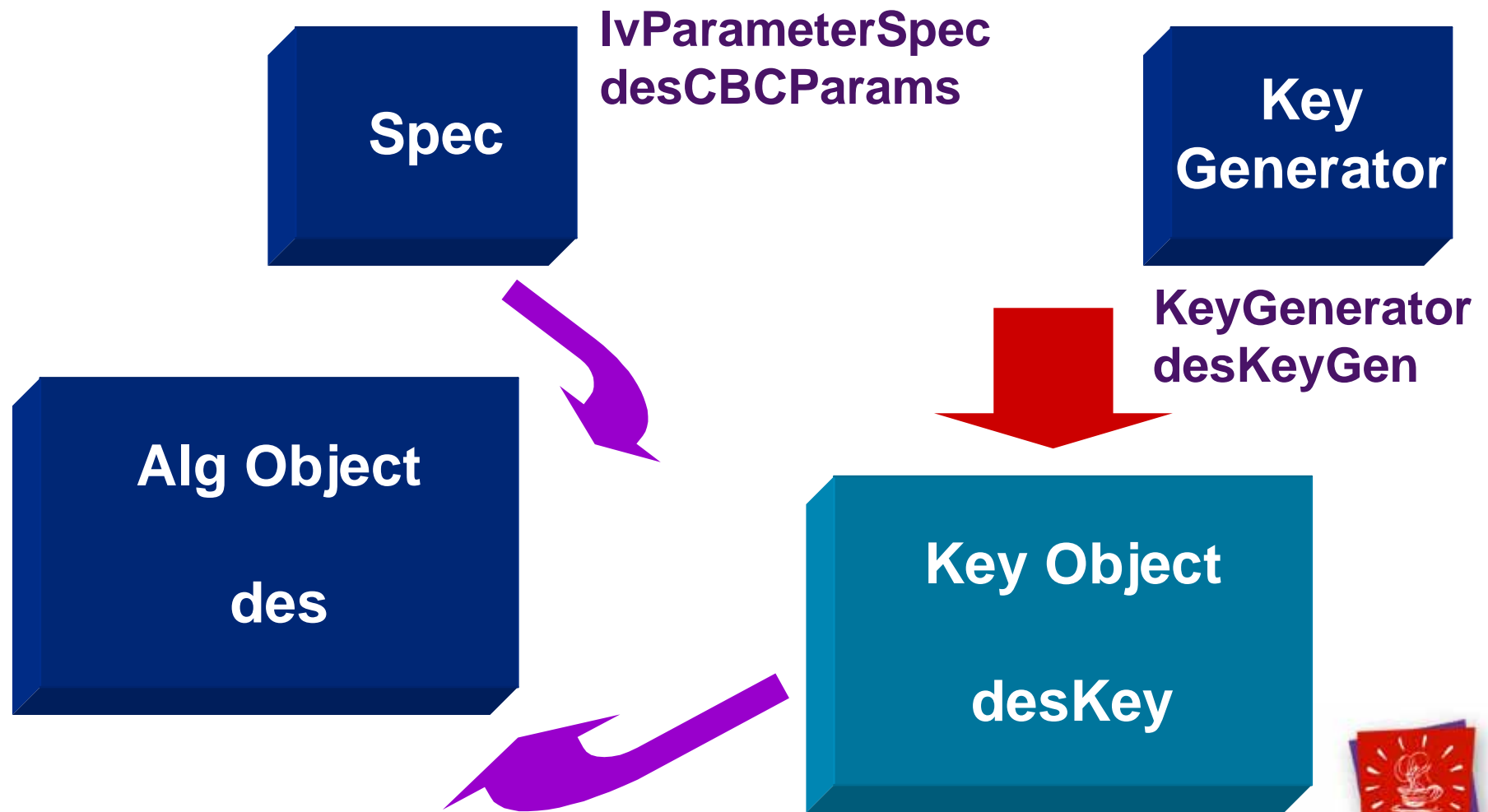
Incidentally

Standard Names

(“DES”, “CBC”, “DiffieHellman”)

file:///D:/jce1.2/doc/guide/API_users_guide.html

JCA/JCE Model



Cipher

Step 2: init

```
des.init  
(Cipher.ENCRYPT_MODE,  
desKey, desCBCParams);
```

Cipher

Step 3: update

```
byte[] dataToEncrypt =  
    new byte[2048];  
byte[] encryptedData =  
    new byte[2056]  
  
int encryptedDataLen = des.update  
    (dataToEncrypt, 0, 2048,  
     encryptedData, 0);
```

Cipher

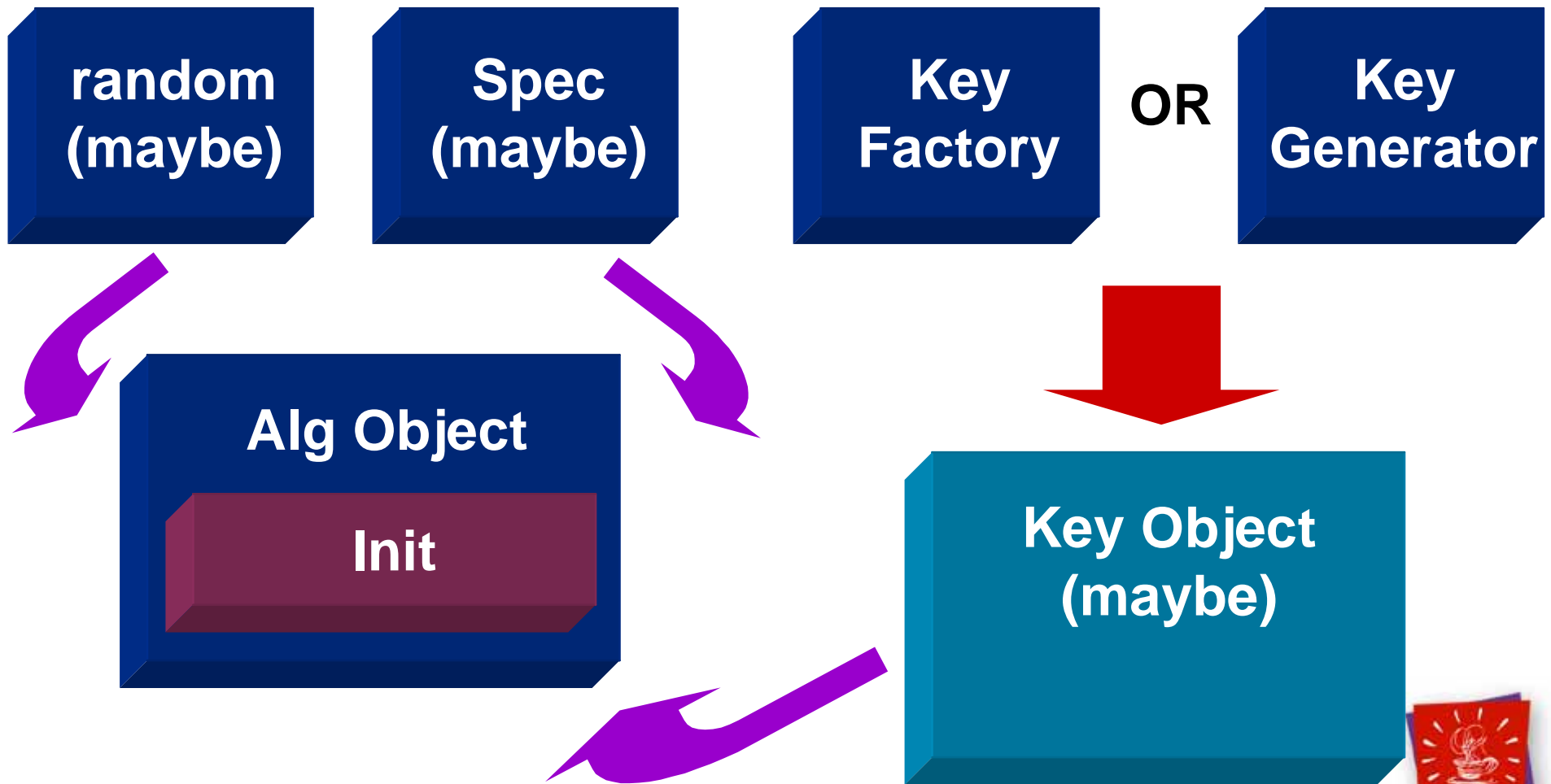
Step 4: final

```
encryptedDataLen +=  
    des.doFinal  
        (encryptedData,  
         encryptedDataLen);
```

The RSA Algorithm

<http://www.rsa.com/rsa/rsamath/index.html>

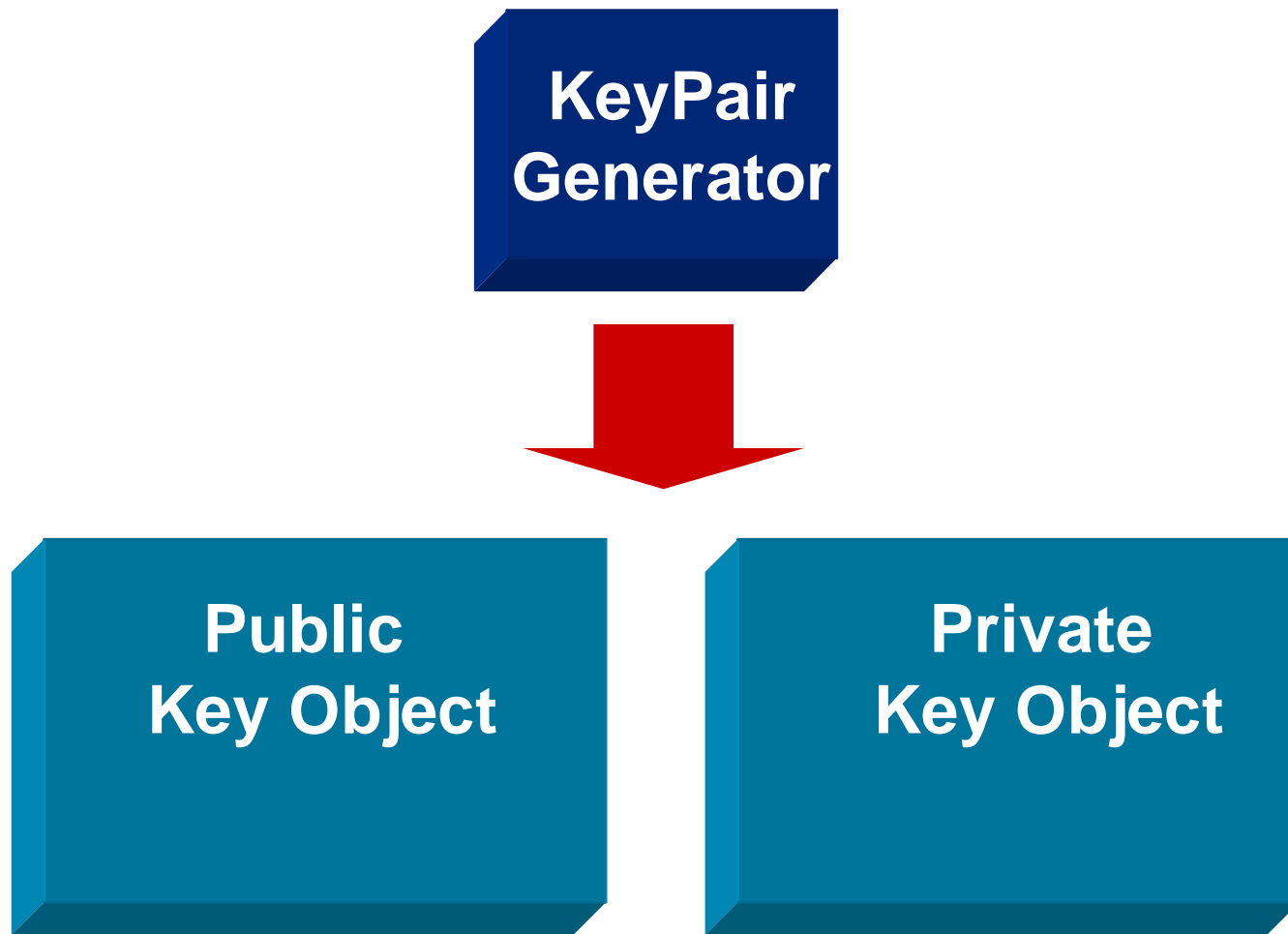
JCA/JCE Model



RSA

- **Algorithm object: Cipher**
- **Spec (not needed)**
- **random (for padding): well-seeded random object**
- **Key Object: PublicKey (built from KeyPairGenerator)**

Generating RSA Key Pairs



KeyPairGenerator

- Algorithm object: KeyPairGenerator
- Spec (not needed)
- random: well-seeded random object
- Key Object (not needed—we're building keys)

KeyPairGenerator

Step 1: getInstance

```
KeyPairGenerator rsaKeyGen =  
KeyPairGenerator.getInstance  
("RSA", "SUNJCE");
```

Doesn't Work

KeyPairGenerator

```
Provider acme = new  
    com.acme.provider.Acme ();  
Security.addProvider (acme);
```

```
KeyPairGenerator rsaKeyGen =  
    KeyPairGenerator.getInstance  
        ( "RSA", "Acme" );
```

Of Course

```
... jdk\jre\lib\java.security
```

```
security.provider.1 =  
    sun.security.provider.Sun
```

```
security.provider.2 =  
    com.acme.provider.Acme
```

KeyPairGenerator

Step 2: init

```
byte[] newSeed =  
    mySeedGenerator.getSeed (20);  
random.setSeed (newSeed);
```

```
rsaKeyGen.initialize  
    (1024, random);
```

 Strength (instead of Spec)

KeyPairGenerator

Step 3, Step 4: Update, Final

```
KeyPair rsaKeyPair =  
    rsaKeyGen.genKeyPair ();
```

```
PublicKey rsaPublic =  
    rsaKeyPair.getPublic ();
```

```
PrivateKey rsaPrivate =  
    rsaKeyPair.getPrivate ();
```



RSA Key Pair

```
byte[] encodedPublic =  
    rsaPublic.getEncoded ();  
  
byte[] encodedPrivate =  
    rsaPrivate.getEncoded ();
```

Getting Key Data

```
Class specClass = Class.forName  
    ( javax.crypto.spec.X509EncodedKeySpec );
```

```
KeyFactory factory =  
    KeyFactory.getInstance  
        ( "RSA", "Acme" );
```

```
X509EncodedKeySpec keySpec =  
    factory.getKeySpec  
        ( rsaPublic, specClass );
```

Getting Key Data

```
byte[] keyData =  
    keySpec.getEncoded ( );
```


RSA

- Algorithm object: Cipher
- ~~• Spec (not needed)~~
- ✓ • random (for padding): well-seeded random object
- ✓ • Key Object: PublicKey (built from KeyPairGenerator)

Cipher

Step 1: getInstance

```
Cipher rsa =  
    Cipher.getInstance  
        ( "RSA", "Acme" );
```

Cipher

Step 2: init

```
rsa.init  
(Cipher.ENCRYPT_MODE,  
rsaPublic, random);
```

Cipher

Step 3: update

```
byte[] keyData =  
    desKey.getEncoded ();  
byte[] encryptedKey =  
    new byte[128]  
  
encryptedKeyLen = rsa.update  
    (keyData, 0, 8, encryptedKey, 0);
```



Cipher

Step 4: final

```
encryptedKeyLen += rsa.doFinal  
    (encryptedKey,  
     encryptedKeyLen);
```

The JCA Model

1. getInstance
2. init
3. update
4. final

Topics

- What is the JCA/JCE
- The JCA Programming Model
- Examples
- Random Number Generation
- [appendix]

Random Numbers

Definition: Pseudo Random Numbers

Numbers produced by a deterministic method that appear to be random

Random Classes

`java.util.Random`



`java.security.SecureRandom`

Instantiating

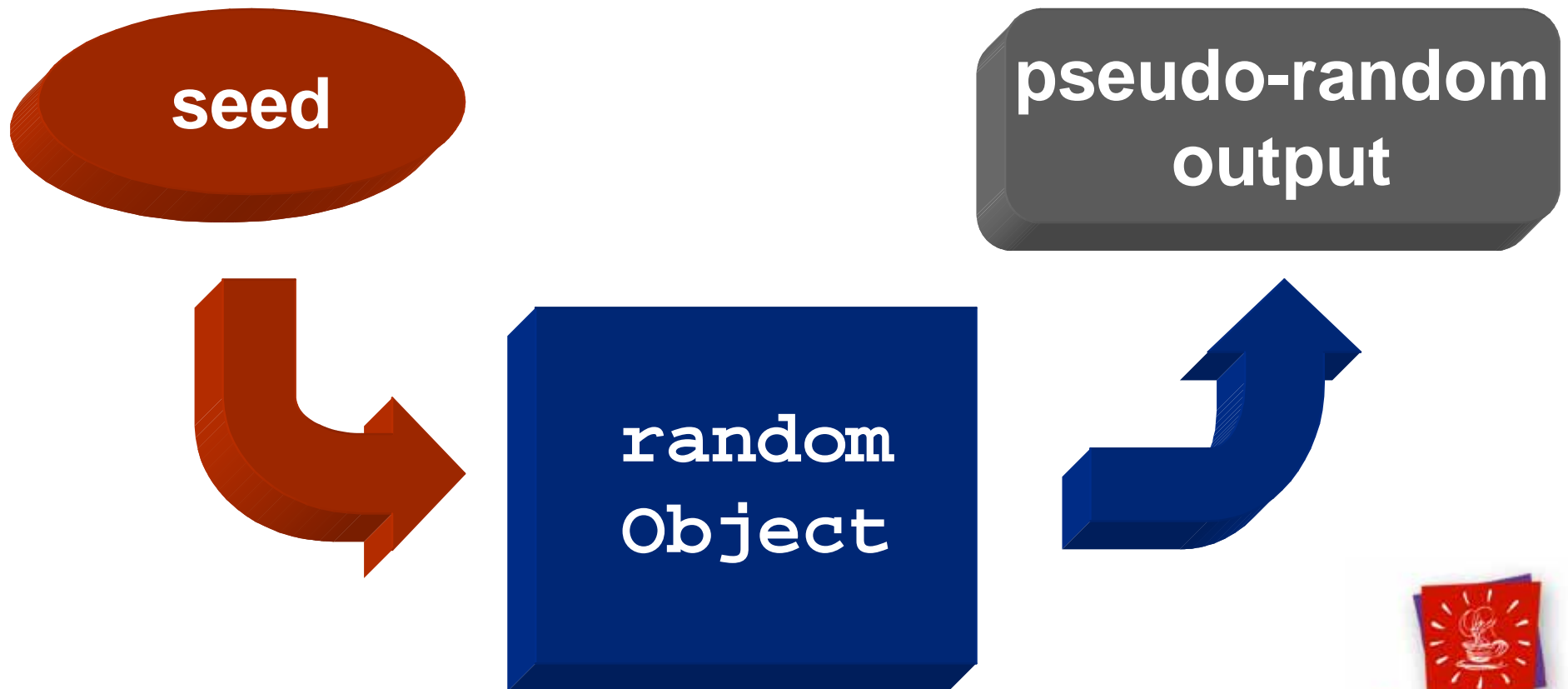
```
SecureRandom random =  
    SecureRandom.getInstance  
        ( "SHA1PRNG" );
```

```
SecureRandom random =  
    new SecureRandom ( );
```

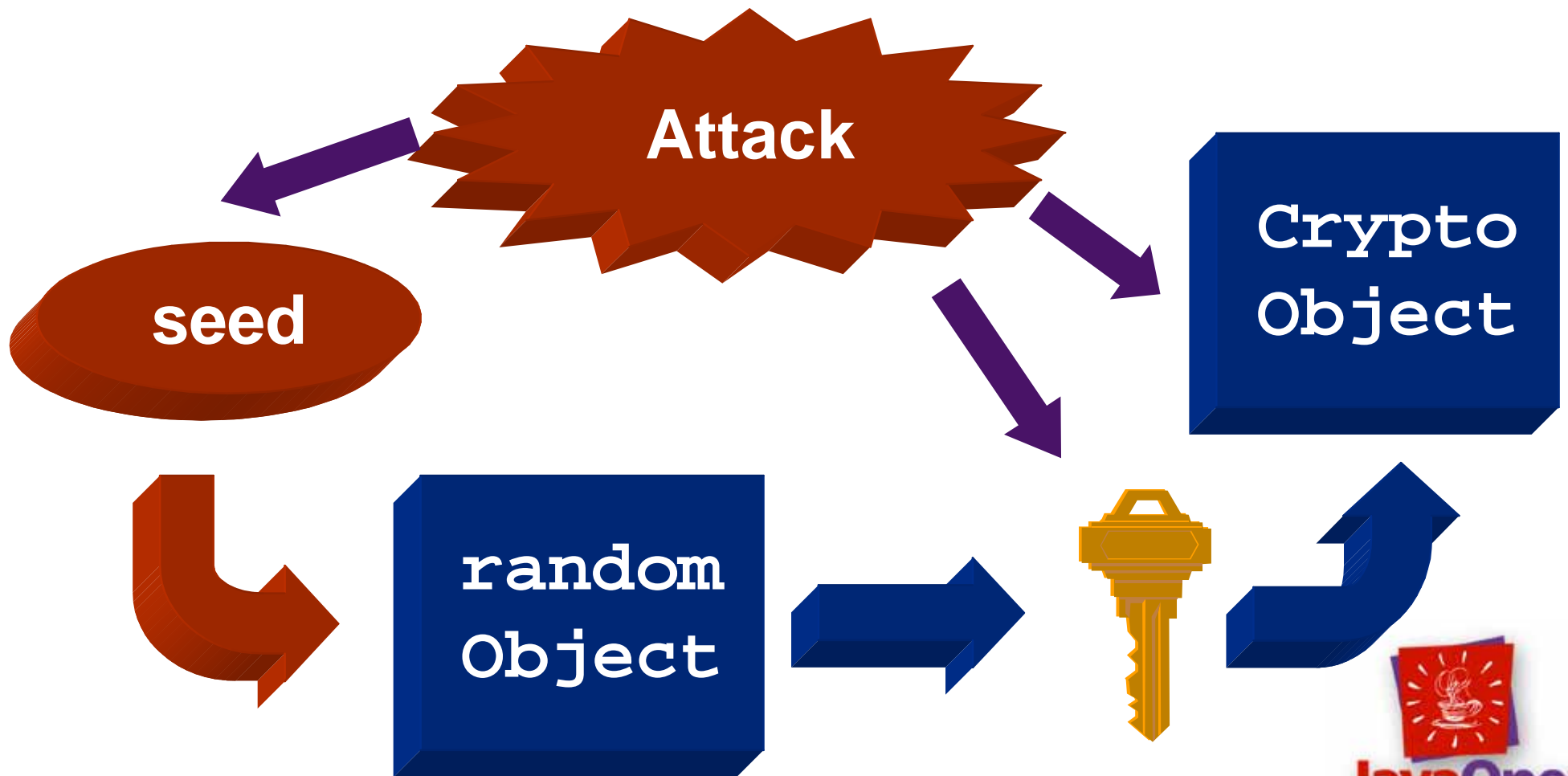
```
SecureRandom random =  
    new SecureRandom (new byte[4]);
```



Random Seeding



Random Seeding



Random Seeding

**The seed should be
as strong as the key**

Quality and Quantity

Quality for unrepeatability

Quantity to increase brute-force time



Random Seeding

Quality = Entropy

Entropy: from order to chaos

Entropy: amount of information

**Memory Statistics, Process Statistics,
Mouse Movement, Keystroke Timing**

<http://www.rsa.com/rsalabs/pubs/updates/bull-1.pdf>

Random Seeding

Quantity

Rule of Thumb

At least

**1 byte of random seed
for each 1 bit of key**

Random Seeding

Why not just use the seed as the key data?

- 1. Some of the seed bits may be known**
- 2. Seed collection may be too time-consuming**
- 3. Seed collection may be platform-dependent**
- 4. Seed bytes may not be pseudo-random**



Autoseeding

SecureRandom class documentation: “We attempt to provide sufficient seed bytes to completely randomize the internal state of the generator (20 bytes). Note, however, that our seed generation algorithm has not been thoroughly studied or widely deployed. It relies on counting the number of times that the calling thread can yield while waiting for another thread to sleep for a specified interval.”

Autoseeding

If you do not seed, the first time the object generates random bytes, it will autoseed

```
byte[] newSeed =  
    random.generateSeed (20);  
random.setSeed (newSeed);
```

LEARN

DRINK

JAVA™
TECHNOLOGY

LIVE PLAY



EAT BREATHE

Using the JCA/JCE

Steve Burnett
RSA Data Security, Inc.
burnett@rsa.com

Appendix

Passwords

Password-Based Encryption (aka PBE)

`PBEKeySpec (char[] password)`

Why not a String?



Passwords

New York Times, Aug. 27, 1997

“Drake discovered his passphrase was being written out to disk by PGP when he decided to scan through the data stored on his disk with Norton Utilities, a common tool for analyzing hard disks. He searched the disk for the phrase and found it written in five different places.”

Overwrite Sensitive Data

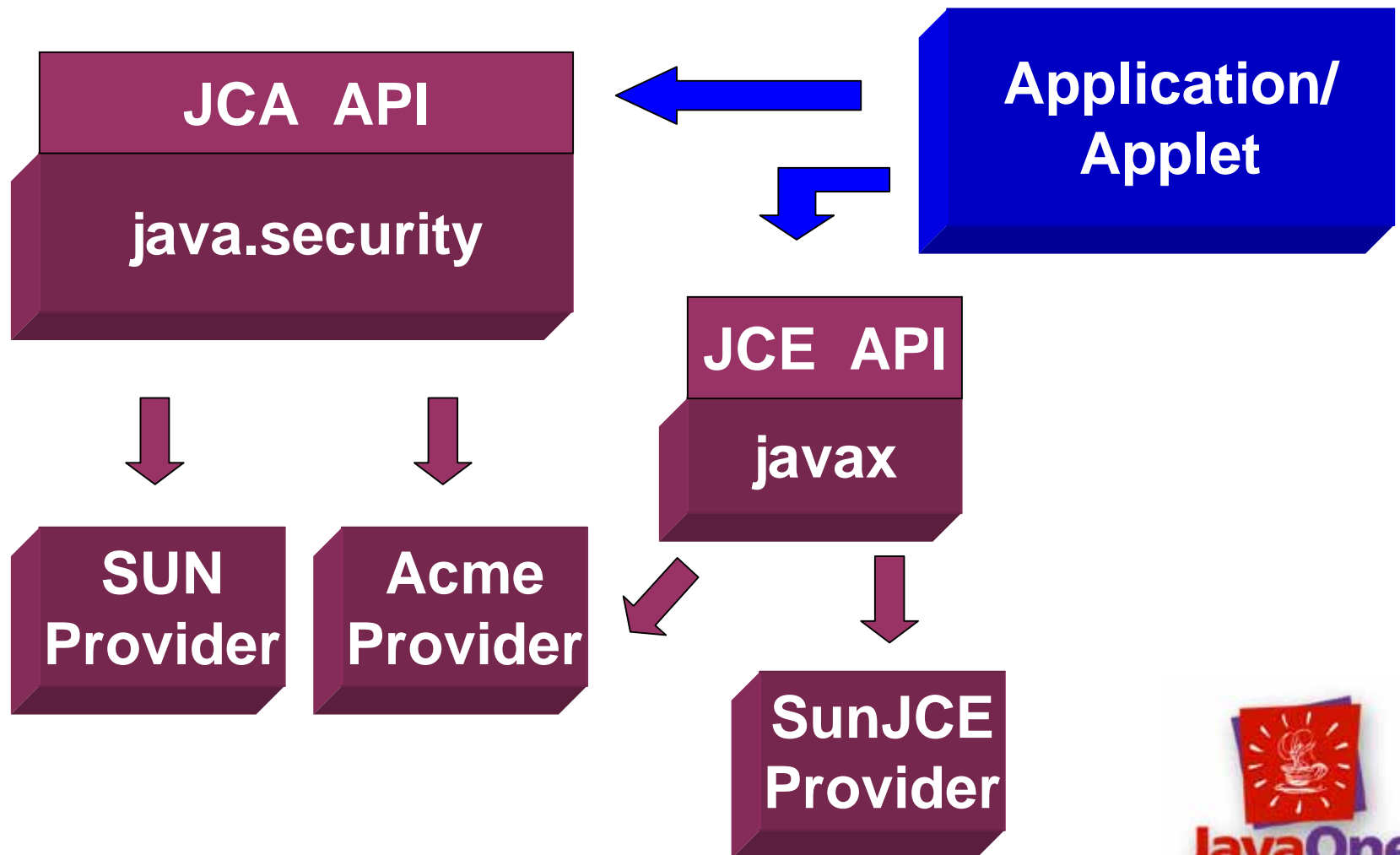
```
byte[] keyData = new byte[16];  
  
random.nextBytes (keyData);  
  
SecretKeySpec keySpec =  
    new SecretKeySpec  
        (keyData, "RC4");  
  
OverwriteClass.overwrite (keyData);
```

Export

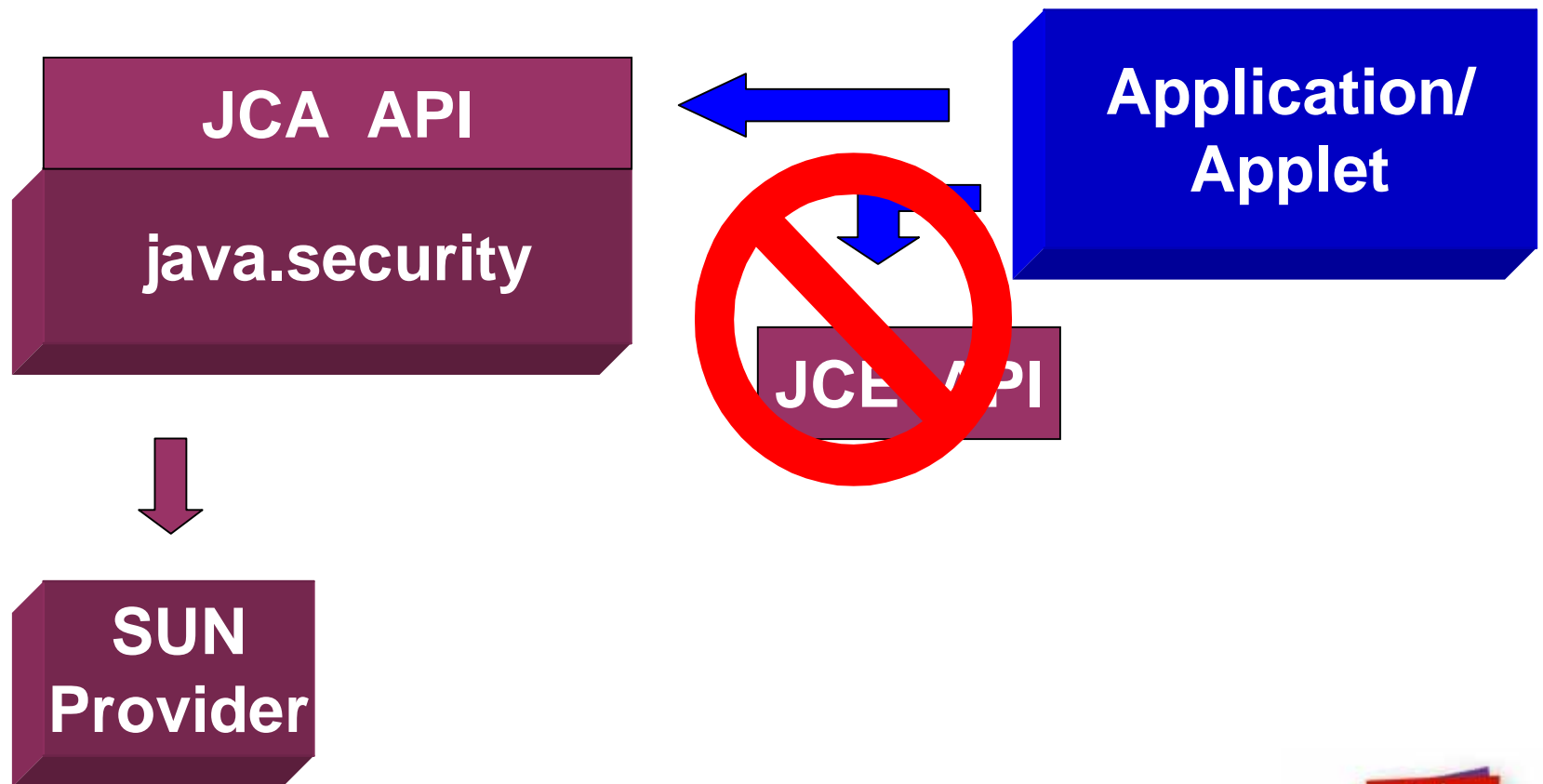
U.S. government does not allow the export of strong encryption except under certain circumstances (e.g. financial or weak crypto)

“Crypto with a hole” is illegal

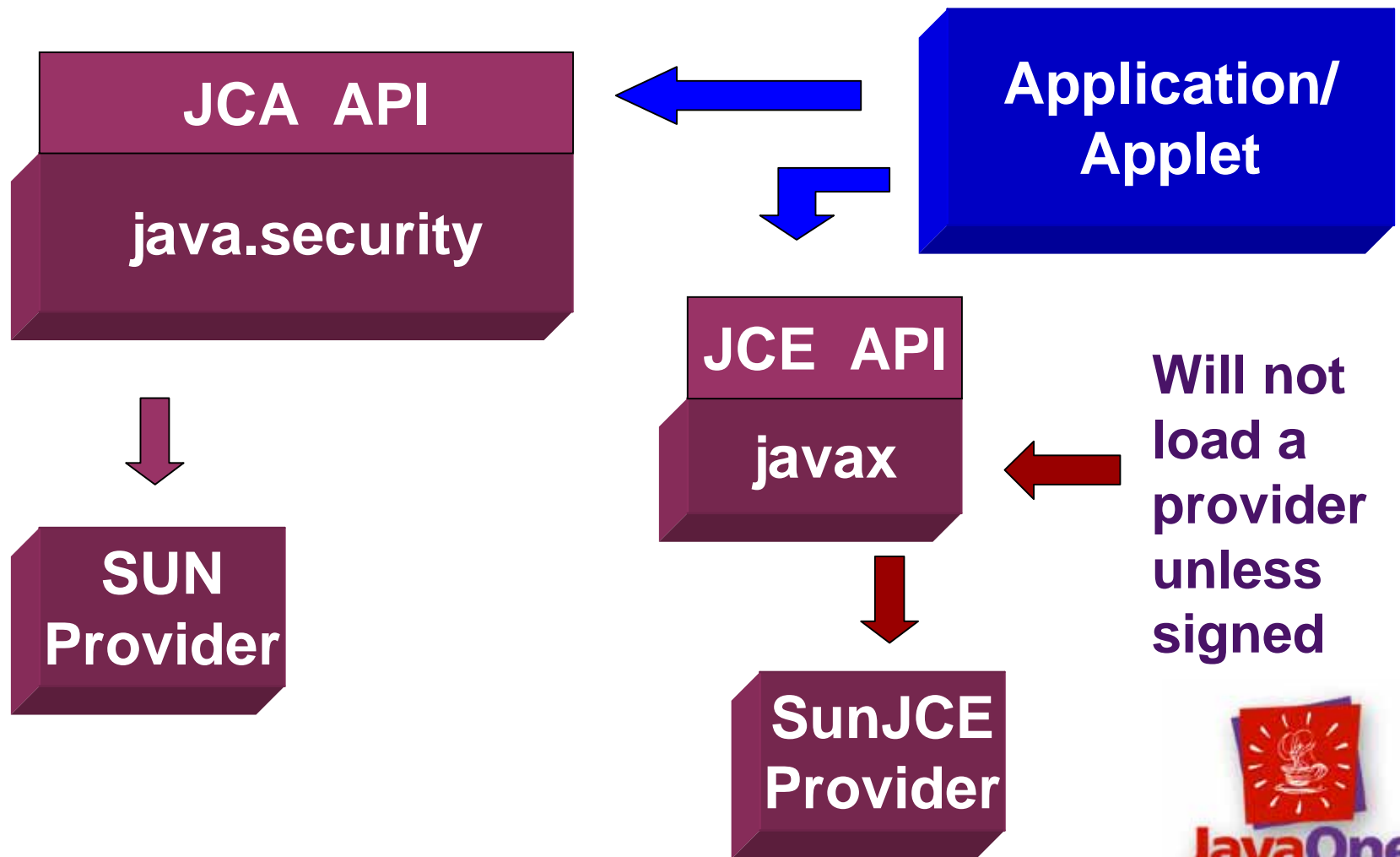
Crypto With a Hole



Crypto With a Hole



Future: Signed Providers



Future: Signed Providers

Microsoft CAPI does this already

U.S. crypto company builds a full strength provider and promises not to export—Java Software (the company) applies a digital signature to the jar file

Any crypto company builds an export strength provider—There's no reason they can't export; Java Software (the company) applies a digital signature to the jar file



Future: Signed Providers

You build an application that calls on the JCE—Either there will be

no crypto provider (no crypto: OK)

unsigned provider (won't load = no crypto: OK)

signed provider (govt knows if it's outside the country, it's weak crypto)

Opening the Envelope

Private Key: `rsaPrivate`

Encrypted DES key: `encryptedKey`

DES encryption parameters: `init vector`
(`IvParameterSpec`)

Encrypted Data: `encryptedData`

Real World

PKCS #7, S/MIME or OpenPGP/MIME message

```
30 11
    06 05
        2b 0e 03 02 07
    04 08
        <init vector>
```

Opening the Envelope

```
Cipher rsa = Cipher.getInstance  
    ("RSA", "Acme");
```

```
rsa.init  
    (Cipher.DECRYPT_MODE, rsaPrivate);
```


Opening the Envelope

```
byte[] decryptedKey = new byte[8];
```

```
int decryptedKeyLen = rsa.update  
    (encryptedKey, 0,  
     encryptedKeyLen,  
     decryptedKey, 0);
```

Opening the Envelope

```
decryptedKeyLen += rsa.doFinal  
    (decryptedKey, decryptedKeyLen);
```

Opening the Envelope

```
SecretKeyFactory factory =  
    SecretKeyFactory.getInstance  
        ( "DES", "SunJCE" );
```

```
SecretKeySpec keySpec =  
    new SecretKeySpec  
        (decryptedKey, 0,  
         decryptedKeyLen, "DES" );
```



Opening the Envelope

```
SecretKey desKey =  
    factory.generateSecret  
        (keySpec);
```

Opening the Envelope

```
Cipher des = Cipher.getInstance  
    ( "DES", "SunJCE" );
```

```
des.init  
    ( Cipher.DECRYPT_MODE, desKey,  
      desCBCParams );
```

Opening the Envelope

```
byte[] decryptedData =  
    new byte[encryptedDataLen];
```

```
int decryptedDataLen = des.update  
    (encryptedData, 0,  
     encryptedDataLen,  
     decryptedData, 0);
```

Opening the Envelope

```
decryptedDataLen += des.doFinal  
    (decryptedData, decryptedDataLen);
```



JavaOneSM

Sun's 1999 Worldwide Java Developer Conference™